Formal Methods in Software Development
Exercise 8 (January 18)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

December 14, 2009

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- A PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the Promela model,
  - a copy of the content of the XSpin "Simulation Output" window for a sample simulation of moderate size (use the button "Save In"),
  - an explicit statement whether the simulation seems adequate and of the result of the verification of each property to be checked,
  - for each property to be checked, the PLTL formula, the definition of the atomic predicates, a screenshot of the verification window with the message "valid/not valid", and (if not valid) a screenshot of the (end of the) counterexample run,
- the source of the Promela model and of the properties verified (as saved from the verification window in a text file).

1 Model Checking an Elevator

Construct a Promela model of the elevator described in Exercise 7 for $N = 3$, simulate it in Spin in order to validate its adequacy, and model check it in order to verify/falsify the following conditions:

1. Always, if the button b_N is on, it does not stay "on" forever.
2. Always, if button b_N or u_N or d_N is "on", the elevator eventually stops at floor N.
3. Always, if button b_N is "on", it stays "on", until the elevator stops at floor N.
4. If infinitely often button b_N or u_N or d_N is "on", the elevator infinitely often stops at floor N.
5. If from some time on no button is "on", then the elevator eventually stops at floor 0 and stays there.
6. Always, if the elevator is at floor N, it will never stop at floor 0.

The answers to all but the last question should be "yes". The counterexample for the last question shows how the elevator returns from floor N to floor 0.

Some hints are given below (see also the attached sample file):

- Define N as a macro and write your model such that it works for arbitrary values of N.
- Define a `proctype Elevator()` for the elevator and start a corresponding process as `run Elevator()`.
- The Promela version of `if (E) C1 else C2` is

```
if
:: E -> C1
:: else -> C2
fi
```

  The Promela version of `if (E) C` is

```
if
:: E -> C
:: else -> skip
fi
```

  The Promela version of `while (E) C` is

```
do
:: E -> C
:: else -> break
od
```

In all cases, if you forget the `else` branch, the system will *deadlock*.

- To check universally quantified conditions, the following macro is useful

```
inline forall(i, from, to, array, result)
{
  result = true;
  i = from;
  do
  :: i < to ->
    if
    :: array[i] == false -> result = false; break;
    :: else -> skip;
    fi;
    i = i+1;
  :: else -> break;
  od;
}
```

  which puts into variable *result* the truth value of $\forall i : from \leq i < to \Rightarrow array[i] = true$. Likewise the attachment contains a macro

```
inline forallExcept(i, from, to, except, array, result)
```

  which computes $\forall i : from \leq i < to \wedge i \neq except \Rightarrow array[i] = true$. Other quantified conditions can be modelled by similar macros.

- Write the system model in a form

```
do :: true ->
  atomic {
    // compute values of conditions in variables
    forall(...);
    forall(...);
    ...
    // perform a system transition
    if
      :: ...
      :: ...
      ...
    fi
  }
od
```

  where you repeatedly first compute the values of all quantified conditions needed in your model and then, based on the results, decide on which transition the system performs.

  Please note that it is always possible to switch a button which is in state "off" to "on". The system must therefore neither terminate nor deadlock.